



第2章 线性表

2.1 线性表的类型定义

2.2 线性表的顺序表示和实现

2.3 线性表的链式表示和实现

2.4 顺序表和链表的比较

2.5 线性表的应用

知识回顾



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

顺序表的特点：以物理位置相邻表示逻辑关系。

顺序表的优点：任一元素均可随机存取。

顺序表的缺点：进行插入和删除操作时，需移动大量的元素。
存储空间不灵活。



2.3 线性表的链式表示和实现

- 链式存储结构
 - 结点在存储器中的位置是任意的，即逻辑上相邻的数据元素在物理上不一定相邻
- 线性表的链式表示又称为非顺序映像或链式映像。



2.3 线性表的链式表示和实现

- 用一组物理位置任意的存储单元来存放线性表的数据元素。
- 这组存储单元既可以是连续的，也可以是不连续的，甚至是零散分布在内存中的任意位置上的。
- 链表中元素的逻辑次序和物理次序不一定相同。



2.3 线性表的链式表示和实现

例：线性表：（赵、钱、孙、李、周、吴、郑、王）

顺序表

存储地址 存储状态

0031	赵
0033	钱
0035	孙
0037	李
0039	周
0041	吴
0043	郑
0045	王

链表

存储地址 存储状态

0001	李	0043
0007	钱	0013
0013	孙	0001
0019	王	NULL
0025	吴	0037
0031	赵	0007
0037	郑	0019
0043	周	0025



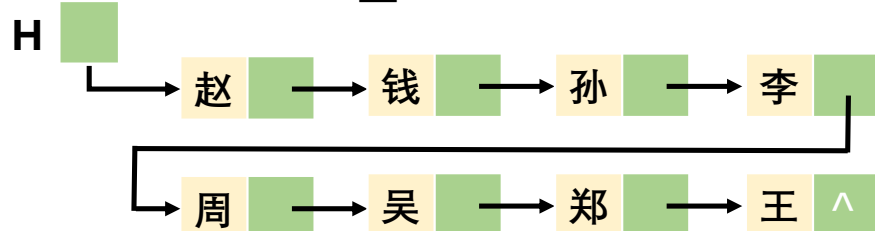
2.3 线性表的链式表示和实现

例：线性表：（赵、钱、孙、李、周、吴、郑、王）

顺序表

存储地址 存储状态

0031	赵
0033	钱
0035	孙
0037	李
0039	周
0041	吴
0043	郑
0045	王



头指针 H

0031

链表

存储地址 数据域 指针域

0001	李	0043
0007	钱	0013
0013	孙	0001
0019	王	NULL
0025	吴	0037
0031	赵	0007
0037	郑	0019
0043	周	0025

结点

链表
单链表

链
指针

单链表是由头指针唯一确定，因此单链表可以用头指针的名字来命名。

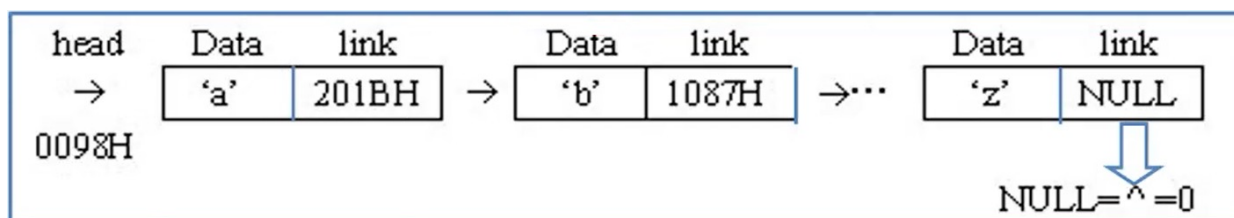


2.3 线性表的链式表示和实现

例：26个英文小写字母的链式存储结构

■ 逻辑结构：(a, b, ..., y, z)

■ 链式存储结构： head → a → b → ... → z → ^



各结点由两个域组成：

数据 | 指针

数据域：存储元素数值数据

指针域：存储直接后继结点的存储位置



2.3 线性表的链式表示和实现

■ 与链式存储有关的术语

1、**结点**：数据元素的储存映像。由数据域和指针域两部分组成

数据域

指针域

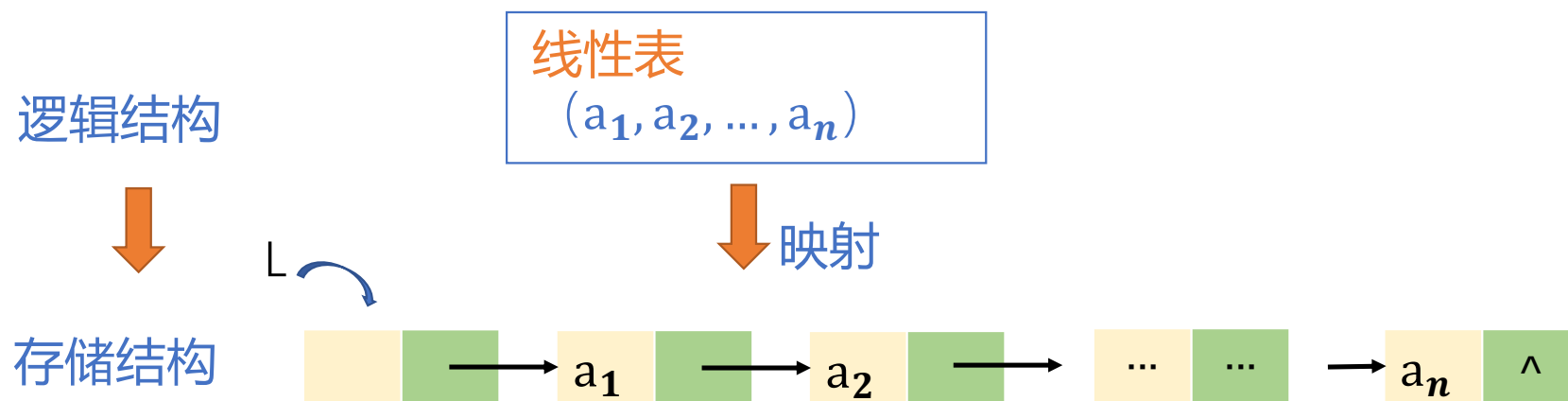
2、**链表**： n 个结点由**指针链**组成一个链表。

它是线性表的链式存储映像，称为线性表的链式存储结构。

2.3 线性表的链式表示和实现



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY



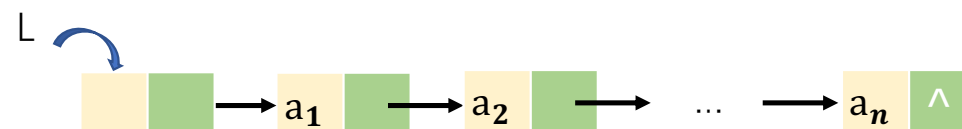
2.3 线性表的链式表示和实现



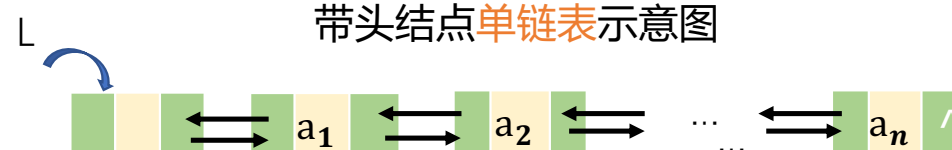
杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

3、单链表、双链表、循环链表：

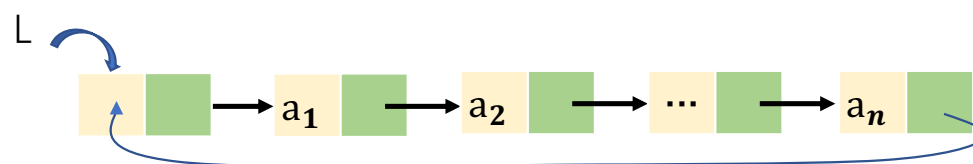
- 结点只有一个指针域的链表，称为**单链表**或线性链表
- 结点有两个指针域链表，称为**双链表**
- 首尾相接的链表称为**循环链表**



带头结点单链表示意图



双链表示意图



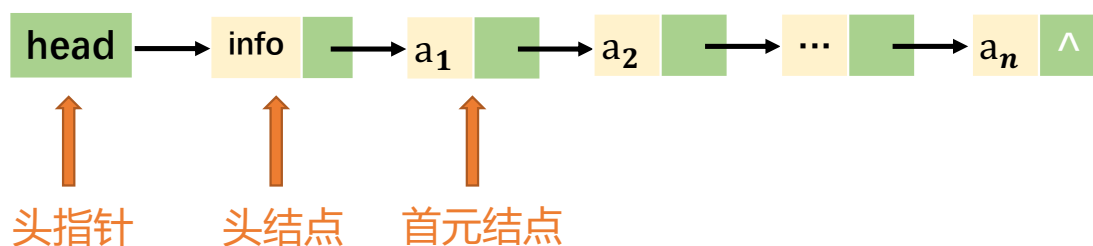
带头结点循环链表示意图

2.3 线性表的链式表示和实现



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

4、头指针、头结点和首元结点：



头指针：是指向链表中第一个结点的指针

首元结点：是指链表中存储第一个数据元素 a_1 的结点

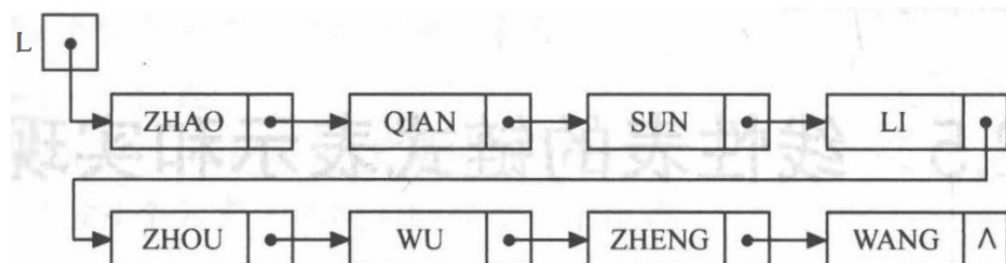
头结点：是在链表的首元结点之前附设的一个结点



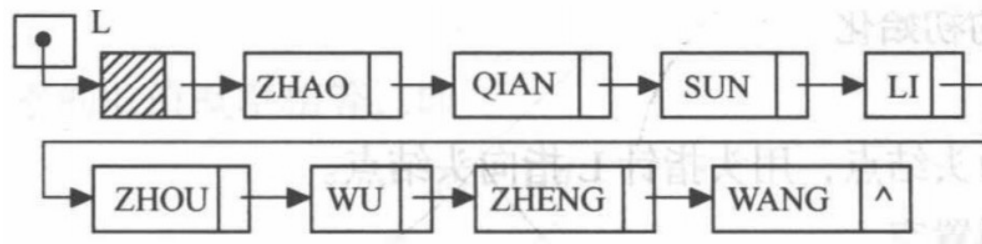
2.3 线性表的链式表示和实现

■ 前面的例子中的例子中的链表的存储结构示意图有以下两种形式：

◆ 不带头结点



◆ 带头结点

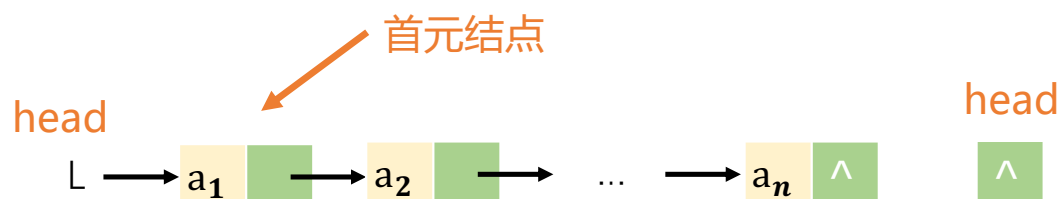




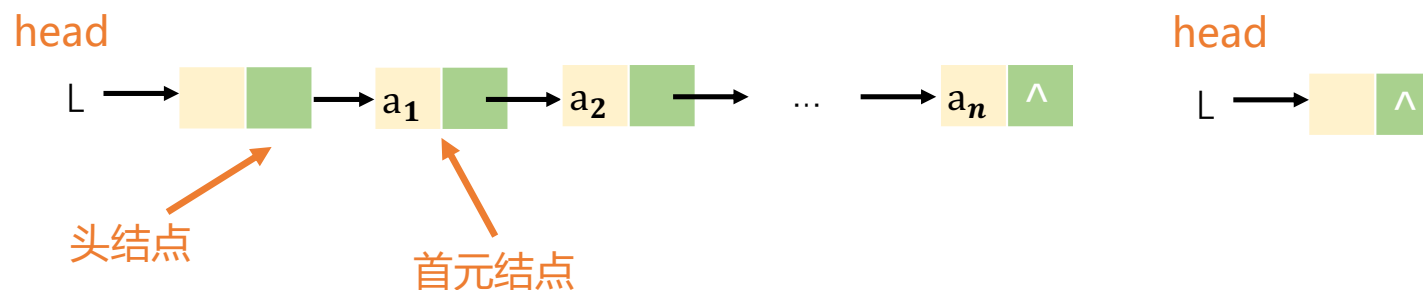
2.3 线性表的链式表示和实现

■ 讨论1: 如何表示空表

- 无头结点时，头指针为空时表示空表



- 有头结点时，当头结点的指针域为空时表示空表





2.3 线性表的链式表示和实现

■ 讨论2：在链表中设置头结点有什么好处？

➤ 1. 便于首元结点的处理

首元结点的地址保存在头结点的指针域中，所以在链表的第一个位置上的操作和其他位置一致，无需进行特殊处理；

➤ 2. 便于空表和非空表的统一处理

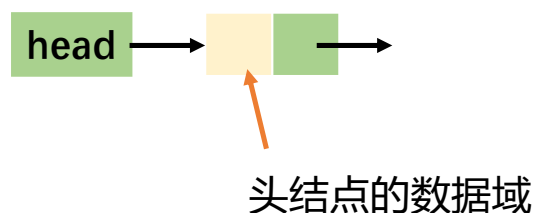
无论链表是否为空，头指针都是指向头结点的非空指针，因此空表和非空表的处理也就统一了。



2.3 线性表的链式表示和实现

■ 讨论3：头结点的数据域内装的是什么？

- 头结点的数据域可以为空，也可存放线性表长度等附加信息，但此结点不能计入链表长度值。





2.3 线性表的链式表示和实现

■ 链表（链式存储结构）的特点

- (1) 结点在存储器中的位置是任意的，即逻辑上相邻的数据元素在物理上不一定相邻。
- (2) 访问时只能通过头指针进入链表，并通过每个结点的指针域依次向后顺序扫描其余结点，所以寻找第一个结点和最后一个结点所花费的时间不等

这种存取元素的方法被称为
顺序存取法



知识总结



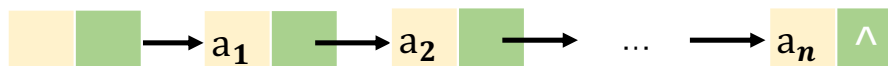
杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

■ 线性表的链式存储结构

- 线性表中数据元素（结点）在存储器中的位置是任意的，即逻辑上相邻的数据元素在物理位置上不一定相邻。

■ 结点：

数据域 指针域



■ 链表： n个结点由指针链组成一个链表

链表是顺序存取的

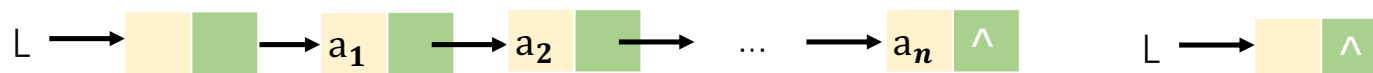
- 单链表：每个结点只有一个指针域
- 双链表：每个结点有两个指针域
- 循环链表：链表结点首尾相接

《 数据结构 》



2.5.1 单链表的定义和表示

■ 带头结点的单链表

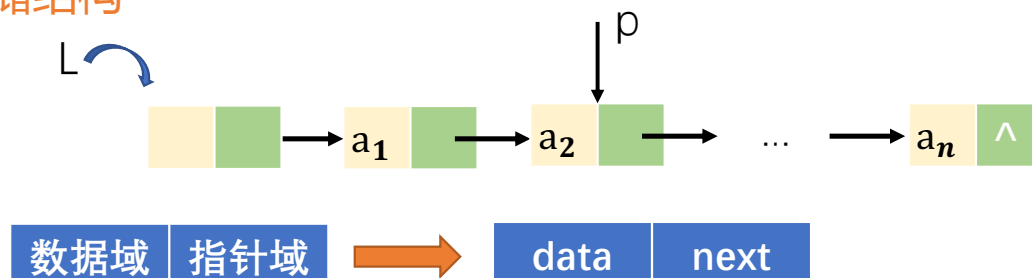


- 单链表是由表头唯一确定，因此单链表可以用头指针的名字来命名，若头指针是L，则把链表称为表L。



2.5.1 单链表的定义和表示

■ 单链表的存储结构



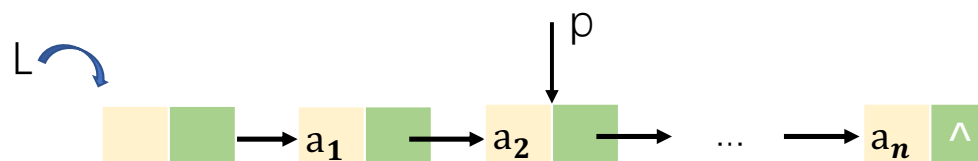
```
typedef struct Lnode{           // 声明结点的类型和指向结点的指针类型
    ElemType    data;           // 结点的数据域
    struct Lnode *next;         // 结点的指针域
} Lnode, *LinkList;           // LinkList为指向结构体Lnode的指针类型
```



2.5.1 单链表的定义和表示

■ 单链表的存储结构

```
typedef struct Lnode{  
    ElemType    data;  
    struct Lnode *next;  
} Lnode, *LinkList;
```



定义链表L: **LinkList L;**

定义结点指针p: **LNode *p;**



LinkList p;

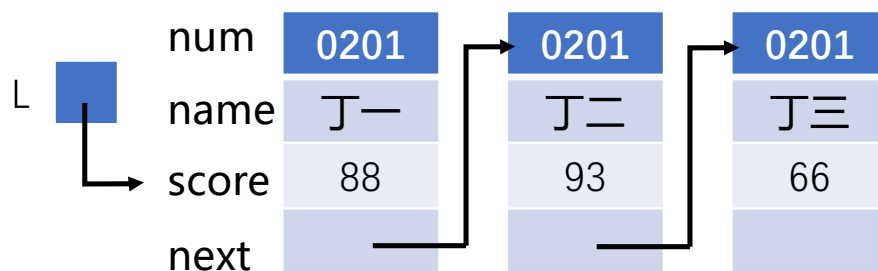


2.5.1 单链表的定义和表示

- 例如，存储**学生学号，姓名，成绩**的单链表结点类型定义如下：

```
typedef struct student {  
    char num[8];           //数据域  
    char name[8];          //数据域  
    int score;              //数据域  
    struct student *next;  //指针域  
}Lnode, *LinkList;
```

LinkList L;



为了统一链表的操作，通常这样定义：

```
typedef struct {  
    char num[8];           //数据域  
    char name[8];          //数据域  
    int score;              //数据域  
}ElemType;  
  
typedef struct Lnode{  
    ElemType data;          //数据域  
    struct Lnode *next     //指针域  
}Lnode, *LinkList;
```



2.5.2 单链表基本操作的实现

■ 单链表的初始化 (算法2.6) (带头结点的单链表)

■ 即构造一个如图的空表



【算法步骤】

(1) 生成新结点作头结点，用头指针 **L** 指向头结点。

(2) 将头结点的指针域置空。

【算法描述】

```
Status InitList_L(LinkList &L){  
    L=(LinkList) malloc (sizeof (Lnode));  
    L->next=NULL;  
    return OK;  
}
```

```
typedef struct Lnode{  
    ElemType    data;  
    struct Lnode *next  
}Lnode, *LinkList;
```



2.5.2 单链表基本操作的实现

补充单链表的几个常用简单算法

■ **【补充算法1】**：判断链表是否为空：

空表：链表中无元素，称为空链表（头指针和头结点仍然在）

【算法思路】判断头结点指针域是否为空

若L是空表



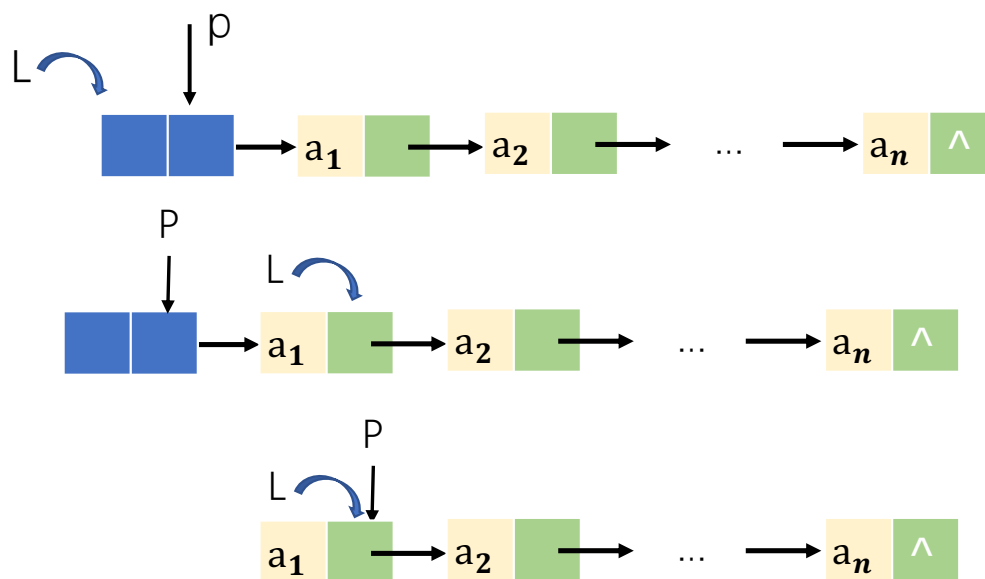
```
int ListEmpty(LinkList L){ //若L为空表，则返回1，否则返回0
    if(L->next) //非空
        return 0;
    else
        return 1;
}
```



2.5.2 单链表基本操作的实现

■ 【补充算法2】：单链表的销毁：链表销毁后不存在

【算法思路】从头指针开始，依次释放所有结点



$p = L;$

$L = L \rightarrow \text{next};$
 $\text{free}(p);$

结束条件: $L = \text{NULL}$
循环条件: $L \neq \text{NULL}$ 或 L



2.5.2 单链表基本操作的实现

■ 【算法】销毁单链表L

```
Status DestroyList_L(LinkList &L){ // 销毁单链表L
    Lnode *p; //或 LinkList p;
    while(L){
        p=L;
        L=L->next;
        free (p);
    }
    return OK;
}
```

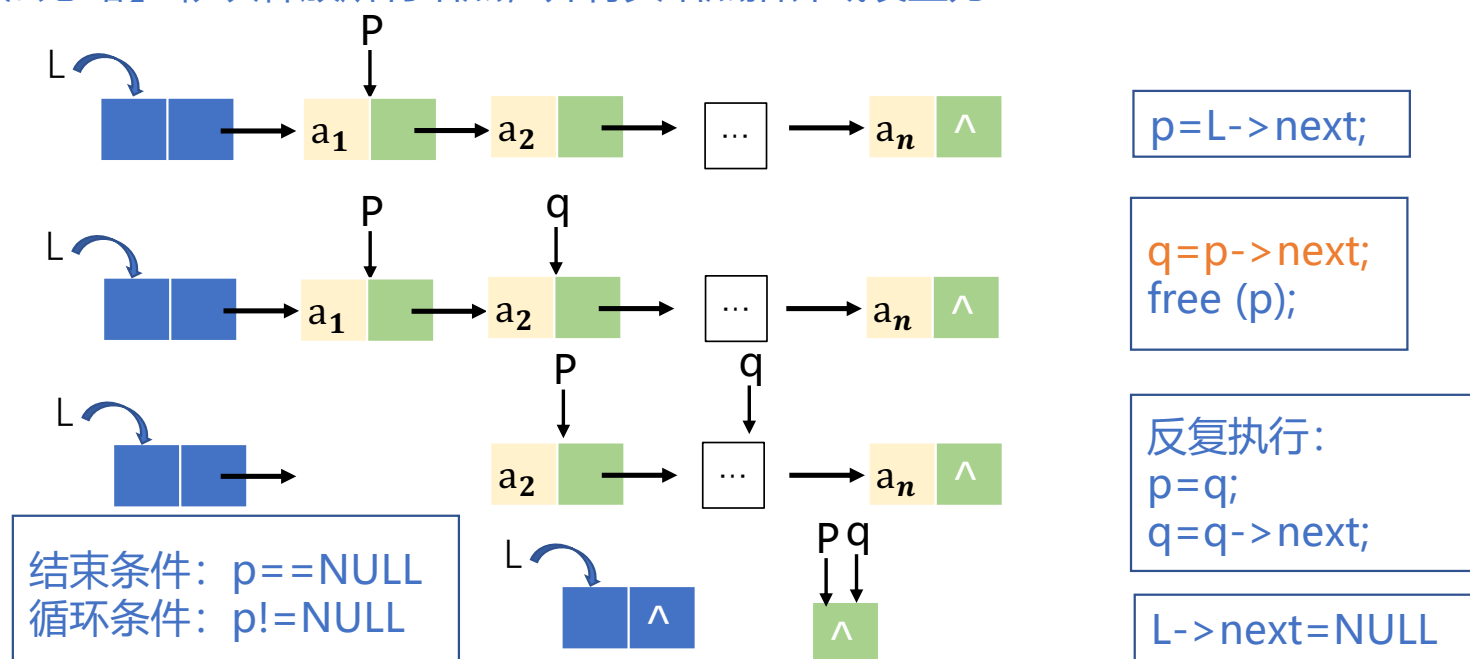


2.5.2 单链表基本操作的实现

■ 【补充算法3】：清空链表：

链表仍存在，但链表中无元素，成为空链表（头指针和头结点仍然在）

【算法思路】 依次释放所有结点，并将头结点指针域设置为空





2.5.2 单链表基本操作的实现

■ 【算法】清空链表L:

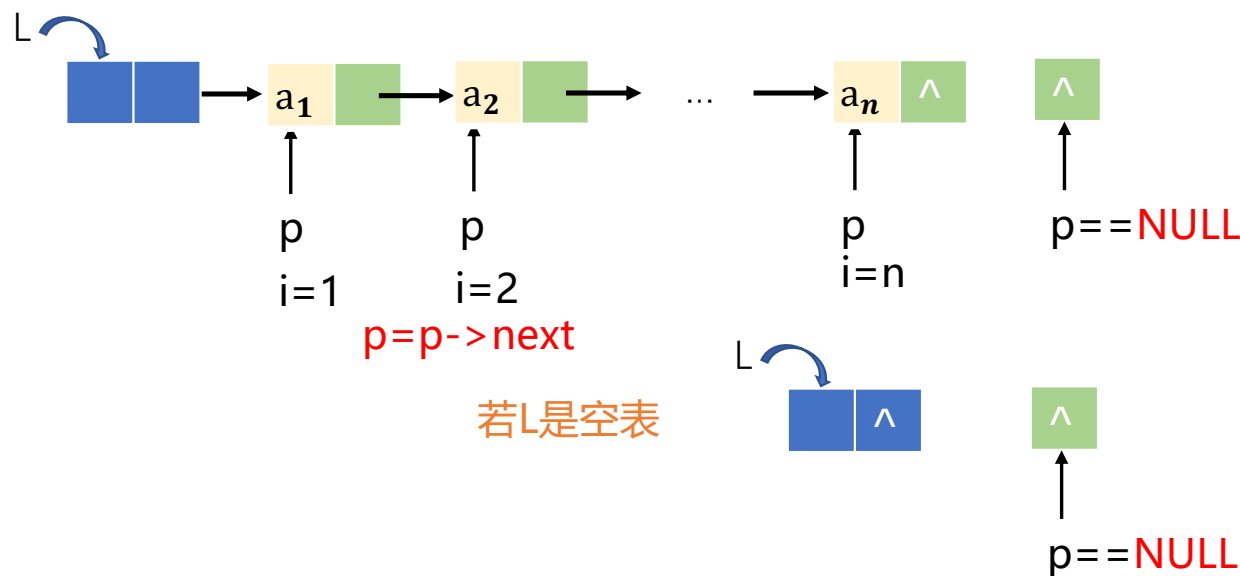
```
Status ClearList(LinkList &L){ // 将L重置为空表
    Lnode *p, *q; //或 LinkList p,q;
    p=L->next;
    while(p){ //没到表尾
        q=p->next;
        free (p);
        p=q;
    }
    L->next=NULL; //头结点指针域为空
    return OK;
}
```



2.5.2 单链表基本操作的实现

■ 【补充算法4】：求单链表的表长：

【算法思路】 从首元结点开始，依次计数所有结点





2.5.2 单链表基本操作的实现

■ 【算法】求单链表L的表长

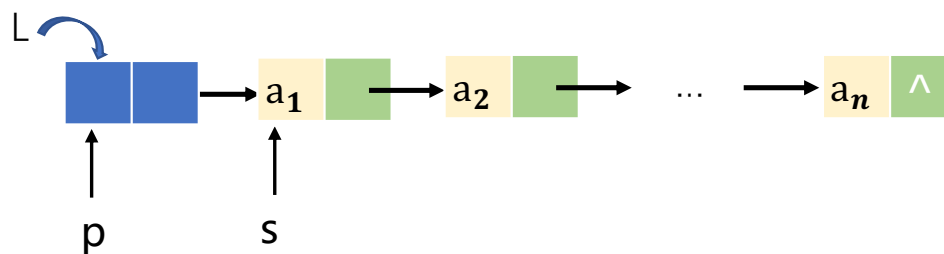
```
int ListLength_L(LinkList L){ // 返回L中数据元素个数
    LinkList p;
    p=L->next; //p指向第一个结点
    i =0;
    while(p){ //遍历单链表, 统计结点数
        i++;
        p=p->next;
    }
    return i;
}
```

阶段总结



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

■ 带头结点的单链表



类型定义:

```
typedef struct LNode {  
    ElemType    data;  
    struct LNode *next;  
} LNode, *LinkList;
```

变量定义:

```
LinkList L;  
LNode *p, *s;
```

重要操作:

```
p=L; //p指向头结点  
s=L->next; //s指向首元结点  
p=p->next; //p指向下一结点
```

阶段总结



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

■ 单链表的基本操作

- 单链表的销毁
- 清空单链表
- 求单链表的表长
- 判断链表是否为空

■ 取值：取单链表中第 i 个元素的内容

■ 查找：

■ 按值查找：根据指定数据获取数据所在位置（地址）

■ 按值查找：根据指定数据获取数据所在位置序号（是第几个元素）

■ 插入：在第 i 个结点前插入新结点

■ 删除：删除第 i 个结点

■ 单链表的建立

■ 头插法

■ 尾插法

《数据结构》

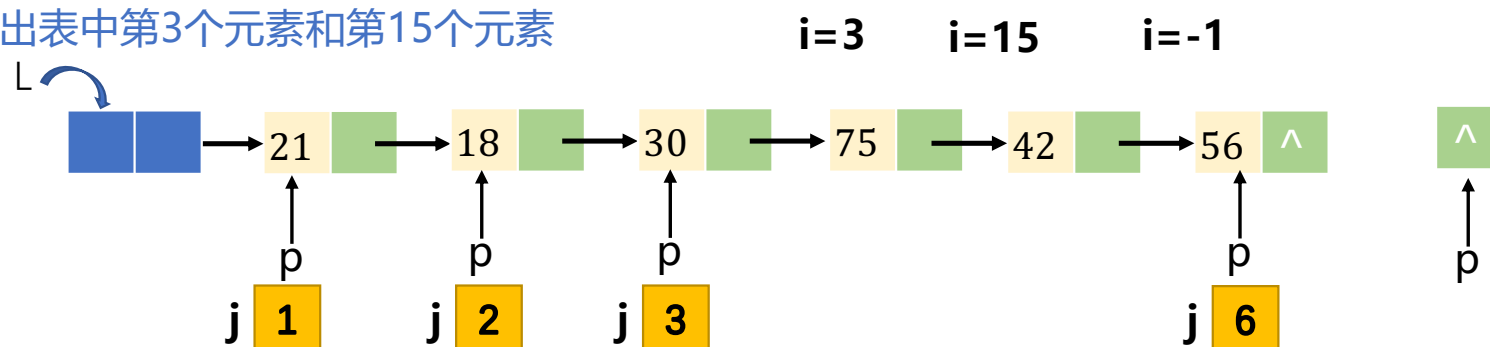


2.5.2 单链表基本操作的实现

【算法2.8】取值——取单链表中第*i*个元素的内容

- 思考：顺序表中如何找到第*i*个元素？ $L \rightarrow \text{elem}[i-1]$

分别取出表中第3个元素和第15个元素



【算法思路】从链表头指针出发，顺着链域next逐个结点往下搜索到第*i*个结点为止。

因此，链表不是随机存取结构



2.5.2 单链表基本操作的实现

【算法步骤】

1. 从第一个结点 ($L \rightarrow next$) 顺链扫描, 用指针 p 指向当前扫描到的结点,
 p 初值 $p = L \rightarrow next$;
2. j 做计数器, 累计当前扫描过的结点数, j 初值为1;
3. 当 p 指向扫描到的下一结点时, 计数器 j 加1;
4. 当 $j = i$ 时, p 所指的结点就是要找的第 i 个结点。



2.5.2 单链表基本操作的实现

【算法描述】

```
Status GetElem_L(LinkList L, int i, ElemType &e){  
    // 获取线性表L中的某个数据元素的内容，通过变量e返回  
    p=L->next; j=1;           //初始化  
    while(p && j<i){           //向后扫描，直到p指向第i个元素或p为空  
        p=p->next;  
        ++j;  
    }  
    if(!p || j>i) return ERROR; //第i个元素不存在  
    e=p->data;                  //取第i个元素  
    return OK;  
} //GetElem_L
```

阶段总结



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

■ 单链表的基本操作

- 单链表的销毁
- 清空单链表
- 求单链表的表长
- 判断链表是否为空

■ 取值：取单链表中第 i 个元素的内容

■ 查找：

- 按值查找：根据指定数据获取数据所在位置（地址）
- 按值查找：根据指定数据获取数据所在位置序号（是第几个元素）

■ 插入：在第 i 个结点前插入新结点

■ 删除：删除第 i 个结点

■ 单链表的建立

- 头插法
- 尾插法

《 数据结构 》

2.5.2 单链表基本操作的实现

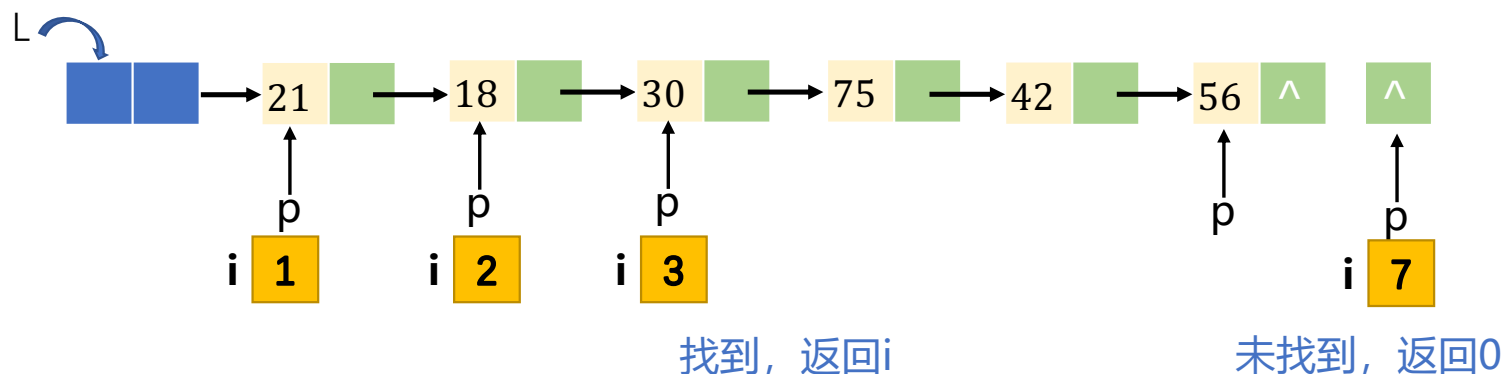


杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

【算法】 按值查找——根据指定数据获取数据所在的位置（该数据的地址）

分别查找值为30和值为15的元素

$e=30$ $e=15$



`p=L->next;`

`p=p->next;`

`p->data!=e;`

《 数据结构 》



2.5.2 单链表基本操作的实现

【算法步骤】

1. 从第一个结点起，依次和e相比较；
2. 如果找到一个其值与e相等的数据元素，则返回其在链表中的“位置”或地址；
3. 如果查遍整个链表都没有找到其值和e相等的元素，则返回0或NULL。



2.5.2 单链表基本操作的实现

【算法描述】

```
Lnode *LocateElem_L(LinkList L, ElemType e){  
    // 在线性表L中查找值为e的数据元素  
    // 找到，则返回L中值为e的数据元素的地址，查找失败返回NULL  
    p=L->next;  
    while(p && p->data!=e){  
        p=p->next;  
    }  
    return p;  
}
```



2.5.2 单链表基本操作的实现

【算法-变化】按值查找——根据指定数据获取该数据位置序号

【算法描述】

```
//在线性表L中查找值为e的数据元素的位置序号
int LocateElem_L(LinkList L, ElemType e){
    // 返回L中值为e的数据元素的位置地址，查找失败返回0
    p=L->next; j=1;
    while(p && p->data!=e){
        p=p->next; j++;
    }
    if(p) return j;
    else return 0;
}
```

阶段总结



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

■ 单链表的基本操作

- 单链表的销毁
- 清空单链表
- 求单链表的表长
- 判断链表是否为空

■ 取值：取单链表中第 i 个元素的内容

■ 查找：

- 按值查找：根据指定数据获取数据所在位置（地址）
- 按值查找：根据指定数据获取数据所在位置序号（是第几个元素）

■ 插入：在第 i 个结点前插入新结点

■ 删除：删除第 i 个结点

■ 单链表的建立

- 头插法
- 尾插法

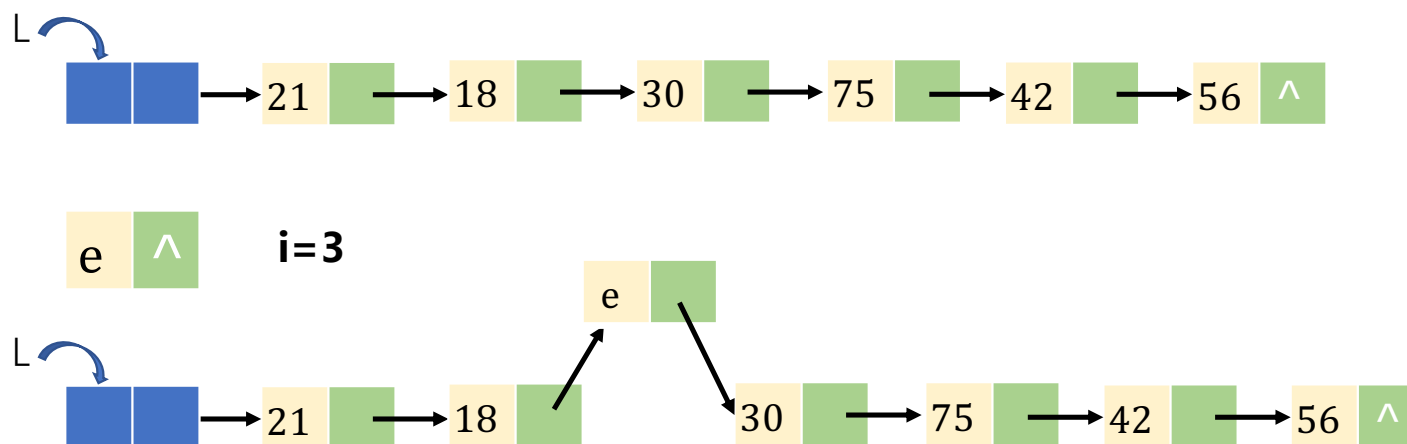
《 数据结构 》

2.5.2 单链表基本操作的实现



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

【算法2.9】插入——在第*i*个结点前插入值为*e*的新结点



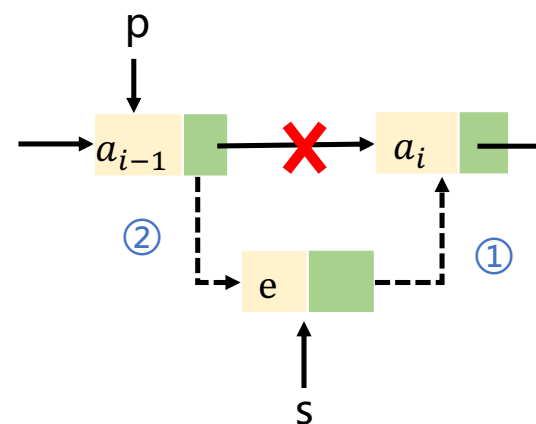


2.5.2 单链表基本操作的实现

【算法步骤】

1. 首先找到 a_{i-1} 的存储位置 p ;
2. 生成一个数据域为 e 的新结点 s ;
3. 插入新结点:

- ① 新结点的指针域指向结点 a_i ;
- ② 结点 a_{i-1} 的指针域指向新结点。



- ① $s \rightarrow \text{next} = p \rightarrow \text{next};$ ② $p \rightarrow \text{next} = s;$

思考：步骤①和步骤②能互换吗？先执行②，后执行①，可以吗？

不可以！会丢失 a_i 的地址



2.5.2 单链表基本操作的实现

【算法描述】

```
//在L中第i个元素之前插入数据元素e
Status ListInsert_L(LinkList &L, int i, ElemType e){
    p=L; j=0;
    while(p && j<i-1){
        p=p->next; ++j;} //寻找第i-1个结点, p指向i-1结点
    if(!p || j>i-1) return ERROR; //i大于表长+1或者小于1, 插入位置非法
    s=(LinkList) malloc (sizeof (LNode));
    s->data=e; //生成新结点s, 将结点s的数据域置为e
    s->next=p->next; //将结点s插入L中
    p->next=s;
    return OK;
} //ListInsert_L
```

阶段总结



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

■ 单链表的基本操作

- 单链表的销毁
- 清空单链表
- 求单链表的表长
- 判断链表是否为空

■ 取值：取单链表中第 i 个元素的内容

■ 查找：

- 按值查找：根据指定数据获取数据所在位置（地址）
- 按值查找：根据指定数据获取数据所在位置序号（是第几个元素）

■ 插入：在第 i 个结点前插入新结点

■ 删除：删除第 i 个结点

■ 单链表的建立

- 头插法
- 尾插法

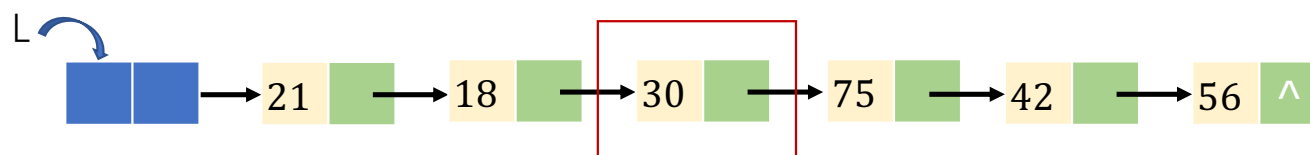
《数据结构》

2.5.2 单链表基本操作的实现

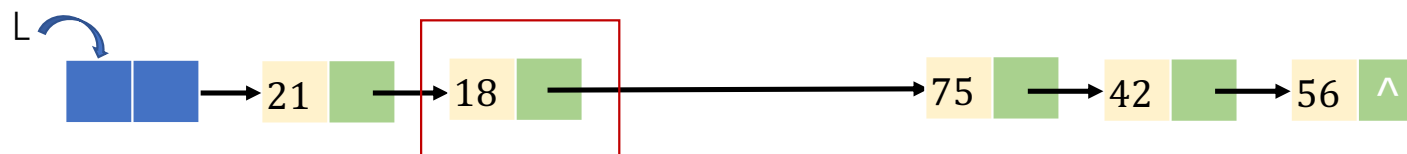


杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

【算法2.10】删除——删除第*i*个结点



$i=3$



《数据结构》

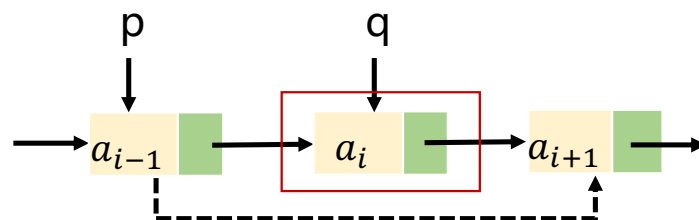


2.5.2 单链表基本操作的实现

【算法2.10】删除——删除第 i 个结点

【算法步骤】

1. 首先找到 a_{i-1} 的存储位置 p ，保存要删除的 a_i 的值；
2. 令 $p \rightarrow \text{next}$ 指向 a_{i+1} ；
3. 释放结点 a_i 的空间



$p \rightarrow \text{next} = p \rightarrow \text{next} \rightarrow \text{next};$



2.5.2 单链表基本操作的实现

【算法描述】

```
//将线性表L中第i个数据元素删除，并由e返回其值
Status ListDelete_L(LinkList &L, int i, ElemType &e){
    p=L; j=0;
    while(p->next && j<i-1){
        p=p->next; ++j; //寻找第i个结点，并令p指向其前驱
    }
    if(!p->next || j>i-1) return ERROR; //删除位置不合理
    q=p->next; //临时保存被删结点的地址以备释放
    p->next=q->next; //改变删除结点前驱结点的指针域
    e=q->data; //保存删除结点的数据域
    free (q); //释放删除结点的空间
    return OK;
} //ListDelete_L
```



2.5.2 单链表基本操作的实现

■ 单链表的查找、插入、删除算法时间效率分析

1. 查找:

2.5.2 单链表基本操作的实现



```
Lnode *LocateElem_L(LinkList L, ElemType e){  
    //在线性表L中查找值为e的数据元素  
    //找到，则返回L中值为e的数据元素的地址，查找失败返回NULL  
    p=L->next;  
    while(p && p->data!=e){  
        p=p->next;  
    }  
    return p;  
}
```



2.5.2 单链表基本操作的实现

■ 单链表的查找、插入、删除算法时间效率分析

1. 查找:

- 因线性链表只能顺序存取，即在查找时要从头指针找起，查找的时间复杂度为 $O(n)$

2. 插入和删除:

- 因线性链表不需要移动元素，只要修改指针，一般情况下时间复杂度为 $O(1)$;
- 但是，如果要在单链表中进行前插或删除操作，由于要从头查找前驱结点，所耗时间复杂度为 $O(n)$ 。

阶段总结



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

■ 单链表的基本操作

- 单链表的销毁
- 清空单链表
- 求单链表的表长
- 判断链表是否为空

■ 取值：取单链表中第 i 个元素的内容

■ 查找：

- 按值查找：根据指定数据获取数据所在位置（地址）
- 按值查找：根据指定数据获取数据所在位置序号（是第几个元素）

■ 插入：在第 i 个结点前插入新结点

■ 删除：删除第 i 个结点

■ 单链表的建立

- 头插法
- 尾插法

《 数据结构 》



2.5.2 单链表基本操作的实现

【算法2.11】建立单链表：头插法——元素插入在链表头部，也叫前插法

1. 从一个空表开始，重复读入数据；
2. 生成新结点，将读入数据存放到新结点的数据域中；
3. 从最后一个结点开始，依次将各结点插入到链表的前端。

■ 例如，建立链表L(a, b, c, d, e)

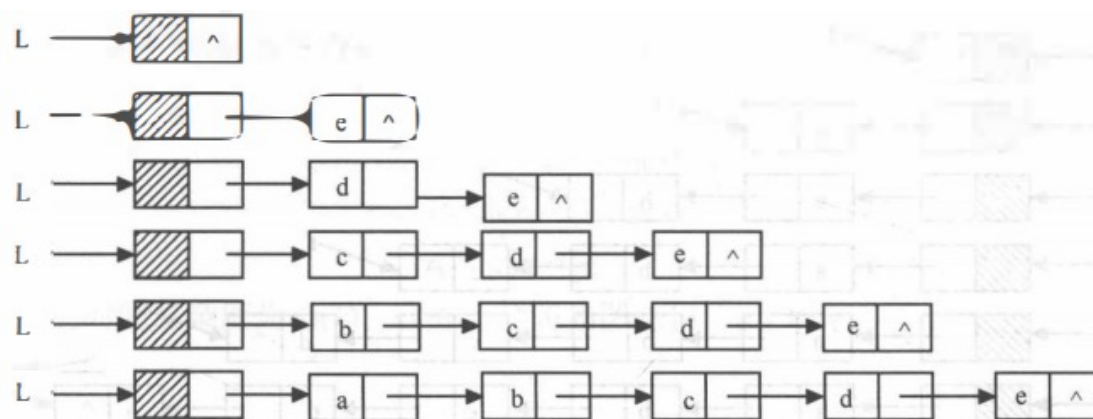


图 2.15 前插法创建单链表

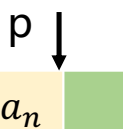


2.5.2 单链表基本操作的实现

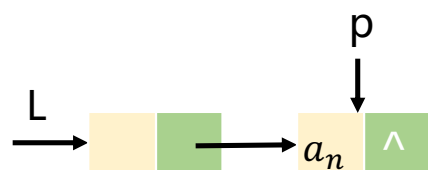


`L=(LinkedList) malloc (sizeof(Lnode));`

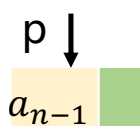
`L->next=NULL;`



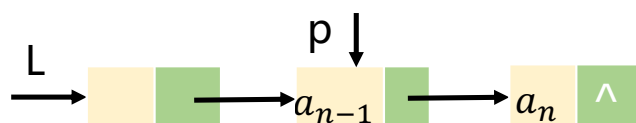
`p =(LinkedList) malloc (sizeof(Lnode)); p->data= a_n ;`



`p->next = L->next;` `L->next=p;`



`p =(LinkedList) malloc (sizeof(Lnode)); p->data= a_{n-1} ;`



`p->next = L->next;` `L->next=p;`



2.5.2 单链表基本操作的实现

【算法描述】

```
void CreateList_H(LinkList &L, int n){  
    L= (LinkList) malloc (sizeof(Lnode));  
    L->next=NULL; //先建立一个带头结点的单链表  
    for(i=n;i>0;--i){  
        p=(LinkList) malloc (sizeof(Lnode)); //生成新结点p  
        scanf(&p->data); //输入元素值  
        p->next=L->next; //插入到表头  
        L->next=p;  
    }  
} //CreateList_H
```

算法时间复杂度: $O(n)$

阶段总结



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

■ 单链表的基本操作

- 单链表的销毁
- 清空单链表
- 求单链表的表长
- 判断链表是否为空

■ 取值：取单链表中第 i 个元素的内容

■ 查找：

- 按值查找：根据指定数据获取数据所在位置（地址）
- 按值查找：根据指定数据获取数据所在位置序号（是第几个元素）

■ 插入：在第 i 个结点前插入新结点

■ 删除：删除第 i 个结点

■ 单链表的建立

- 头插法
- 尾插法

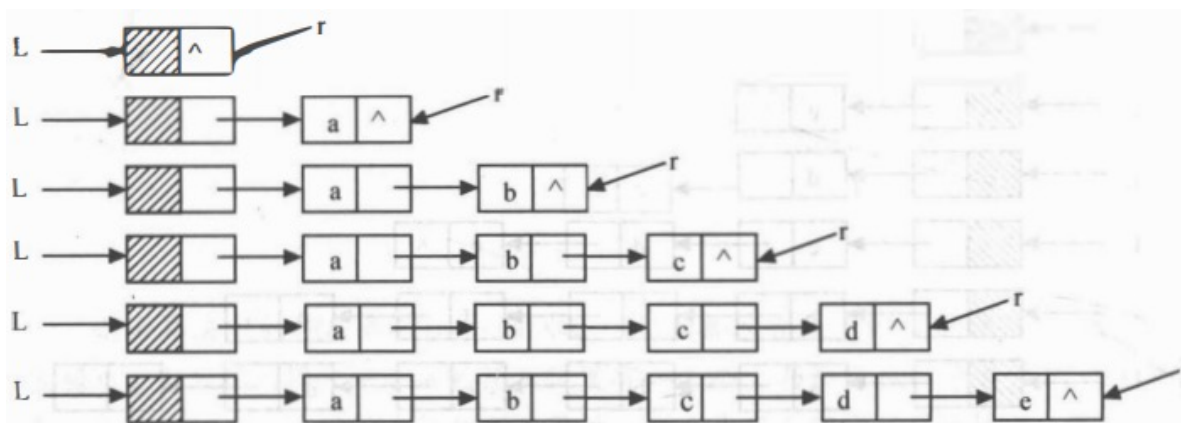
《 数据结构 》



2.5.2 单链表基本操作的实现

【算法-变化】 建立单链表：尾插法——元素插入在链表尾部，也叫后插法

1. 从一个空表L开始，将新结点逐个插入到链表尾部，尾指针r指向链表的尾结点；
2. 初始时，r同L均指向头结点。每读入一个数据元素则申请一个新结点，将新结点插入尾结点后，r指向新结点。



```
p->data = ai;  
p->next = NULL;  
  
r->next = p;  
  
r = p;
```




2.5.2 单链表基本操作的实现

【算法描述】

```
//正位序输入n个元素的值，建立带头结点的单链表L
void CreateList_R(LinkList &L, int n){
    L= (LinkList) malloc (sizeof(Lnode));
    L->next=NULL;
    r=L;          //尾指针r指向头结点
    for(i=0; i<n; ++i){
        p=(LinkList) malloc (sizeof(Lnode)); //生成新结点p
        scanf(&p->data);    //输入元素值
        p->next=NULL;
        r->next=p;    //插入到表尾
        r=p;    //r指向新的尾结点
    }
} //CreateList_R
```

算法的时间复杂度是 $O(n)$

课后题



杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

【题】已知L 是无表头结点的单链表，且P 结点既不是首元结点，也不是尾元结点，试从下列提供的答案中选择合适的语句序列。

- a. 在P 结点后插入S 结点的语句序列是____(4)(1)____
- b. 在P 结点前插入S 结点的语句序列是____(7)(11)(8)(4)(1)____
- c. 在表首插入S 结点的语句序列是____(5)(12)____
- d. 在表尾插入S 结点的语句序列是____(9)(1)(6)____

(1) P->next=S;
(2) P->next=P->next->next;
(3) P->next=S->next;
(4) S->next=P->next;
(5) S->next=L;
(6) S->next=NULL;

(7) Q=P;
(8) while(P->next!=Q) P=P->next;
(9) while(P->next!=NULL) P=P->next;
(10) P=Q;
(11) P=L;
(12) L=S;
(13) L=P;

《 数据结构 》